AD-A100 662 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 12/1
STATISTICAL MEASURES OF SOFTWARE RELIABILITY,(U)
OCT 80 R A DEMILLO, F G SAYWARD N00014-79-C-0231
UNCLASSIFIED GIT-ICS-80/10 NL

1 OF 1
AD A
100662

END
DATE
FILMED
7-81
DTIC

School of

Information and Computer Science

GEORGIA INSTITUTE OF TECHNOLOGY

(1)

(14) GIT-ICS-80/10

(6) STATISTICAL MEASURES OF SOFTWARE RELIABILITY.+

Richard A. DeMillo*

Frederick G. Sayward**

(12) 24

Oct~~ober, 19~~80

*School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

**Computer Science Department
Yale University
New Haven, Conn. 06520

+Work supported in part by Office of Naval Research, Grant (15) N00014-79-C-0231

4 1 0 4 4

STATISTICAL MEASURES OF SOFTWARE RELIABILITY

Richard A. DeMillo
Georgia Institute of Technology

Frederick G. Sayward
Yale University

I. INTRODUCTION

Estimating program reliability presents many of the same problems as measuring software performance and cost: the central technical issue concerns the existence of an independent objective scale upon which may be based a qualitative judgement of the ability of a given program to function as intended in a specified environment over a specified time interval. Several scales have already been proposed. For example, a program may be judged reliable if it has been formally proved correct [1], if it has been run against a valid and reliable test data set [2], or if it has been developed according to a special discipline [3]. While these concepts may have independent interest, they fail to capture the most significant aspect of reliability estimation as it applies to software: most software is unreliable by these standards, but the degree of unreliability is not quantified. A useful program which has not been proved correct is unreliable, but so is, say, the null program (unless by some perversity of specification the null program satisfies the designer); an operationally meaningful scale of reliability should distinguish these extremes.

What is needed is a measure R(t) which, for a given piece of software (i.e., a system, subsystem, program, or program module) gives an index of operational reliability in the time interval [0,t]. The most commonly proposed such index is the reliability function of traditional reliability theory [4,5,6]:

R(t) = "probability" of survival at time t.

In the sequel, we will sketch the outlines of the traditional theory that is most relevant to software reliability estimation, give a brief critical analysis of the use of the traditional theory in measuring reliability, and describe another use of the R(t) measure which we believe more closely fits the intuitive requirements of the scale we asked for above.

2. THE STATISTICAL THEORY

R(t) is to be interpreted as the probability of satisfactory performance of the system in the time interval [0,t]. It is an underlying assumption that satisfactory performance at time t implies satisfactory performance in the interval [0,t]. A second assumption is that the form of the theory does not change from system to system; in particular, it should not matter whether one estimates R(t) for a total system, a subsystem, or a single component.

The second assumption suggests the following analysis of complex systems. Let R_i be the reliability function for the ith component of the system and define a random variable x_i for each component as follows.

$$x_i = \begin{cases} 1, & \text{if } R_i(t) \geq a_i \\ 0, & \text{otherwise} \end{cases}$$

The performance of the system is then determined by a 0-1 valued function

$$\phi(x_1,\ldots,x_n).$$

Simple examples of such performance functions are the functions for series systems,

$$\phi(x_1,\ldots,x_n) = x_1x_2\ldots x_n = \min\{x_i | i \leq n\},$$

and for "parallel" systems,

$$\phi(x_1,\ldots,x_n) = 1-(1-x_1)\ldots(1-x_n) = \max\{x_i | i \leq n\}.$$

For more complex situations, one may place additional requirements on the performance function; for instance, the function may have to be coherent:

$$\text{for all } i,\ x_i \le y_i \implies \phi(y_1,\ldots,y_n) \ge \phi(x_1,\ldots,x_n).$$

The reliability function for the entire system is then

$$R = \text{Prob}\{\phi(x_1,\ldots,x_n)=1\}.$$

For extremely simple models of subsystems failure (e.g., fixed independent probabilities of satisfactory performance) such an analysis gives the combinatorial probabilities of success/failure in a very handy form.

For realistically complex systems, however, R(t) is determined by probability distribution which draw their properties from observable parameters of the system. For computer programs, as well as many other systems, it is inconvenient to estimate satisfactory performance directly. Instead one uses the observed failures of the system. Let F(t) be the failure distribution of the system and let f be the corresponding probability density function. Then

$$R(t) = 1-F(t) = \int_t^\infty f(y)dy.$$

Often, the reliability distribution is more conveniently expressed in terms of the failure rate (or hazard rate). Let B(t,h) represent the conditional probability of failure by time t+h given survival at time t:

$$r(t) = \lim_{h\to 0}\{B(t,h)/h\} = f(t)/R(t)$$

The failure rate determines the reliability since

$$1-F(t) = R(t) = \exp(-\int_0^t r(y)dy).$$

One obtains a reliability function for a given system by a variety of paths: nontechnical considerations like mathematical tractability, empirical observations, or theoretical analysis of more fundamental properties of the system which is subject to failure.

The exponential distribution. The exponential distribution is exactly characterized by those systems which have constant failure rate.

$$R(t) = \exp(-\theta t)$$

$$r(t) = \theta\exp(-\theta t)/\exp(-\theta t) = \theta.$$

The exponential distribution is the most widely used of the reliability models, even when there is slight evidence to justify its use. It is, of course, among the most tractable of the statistical models, but it can lead to serious errors when the underlying distribution corresponds to failures that do not occur randomly or which depend on the past history of the system. The popularity of the exponential distribution (and the tendency to observe it in complex systems) is no doubt due in part to the following fact. Let R_i be the reliability of the ith component and let $R = \Pi R_i$. If there are N components in the system, define $\bar{r}(N)$ to be $r_1(0)+\ldots+r_N(0)$. Then R tends to be an exponential distribution:

$$R(t) \to \exp(-\bar{r}(N)t) \text{ as } N \to \infty,$$

whenever three technical conditions are met:

1. each failure rate grows as

   $r_i(t) = r_i(0) + a_i t^{\theta}$

   as $t \to 0$.

2. $r(N) \to \infty$ as $N \to \infty$.

3. $r(N)(a_i+\ldots+a_N)$ is bounded by a fixed constant.

These conditions are however not easy to satisfy, and several of the most common distributions of reliability theory fail one or more of the restrictions.

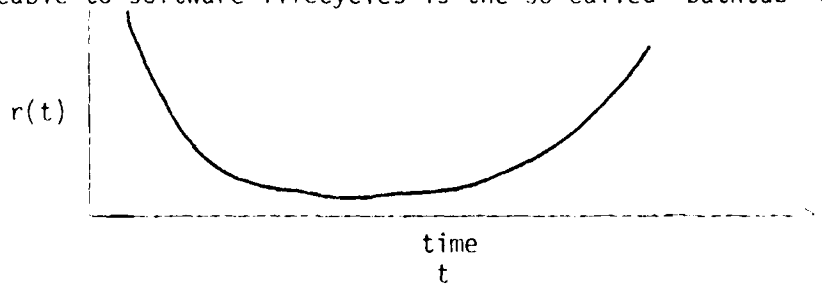
The Weibull distribution. In an attempt to circumvent the more serious deficiencies of the exponential distribution, one is led to a family of distributions in which the failure rate is a function of time.

$$R(t) = \exp(-t^a/\theta),$$

$$r(t) = at^{a-1}/\theta.$$

Obviously, the failure rate is increasing or decreasing depending on the value of a. The properties of the R(t) distribution for certain choices of a and θ make it a good descriptor of mechanical "breakdown" phenomena, such as failures due to metal fatigue or other failures due to stress. The Weibull distribution varies wildly from the exponential distribution in its statistical properties (compare, for example, the nth moments of both) and thus cannot be used interchangeably with predictable results. A special case of the Weibull distribution called the Rayleigh distribution is common in software reliability studies. This class of distributions fails to satisfy restrictions (1) and (3) above, so that it is difficult to argue that interconnections of Rayleigh-distributed components may behave with simpler failure rates.

Truncated normal distribution. The failure rate that is most commonly cited as applicable to software lifecycles is the so-called "bathtub" curve



As opposed to the Weibull-type distributions, the normal distributions model systems which "wear out", e.g., light bulbs. Rather than dealing directly with the reliability functions which may only be applicable to a portion of the system life cycle, it is often convenient to combine several failure rates to model modes of failure which apply to specific stages of development. The wear-out models are particularly attractive in this regard. During the initial stages of operation of a large number of copies of identical systems, there is a high incidence of immediate failures ("infant mortality"). This initiation process is followed by a relatively stable steady-state region until, near the end of the useful lifetime of the systems, the incidence of failures rises.

If $\phi(x)$ is the probability density function for a normal distribution and $\Phi(x)$ is the corresponding distribution function, then even though the expression for R(t) involves an integral that cannot be expressed in closed form, the failure rate is easily expressed:

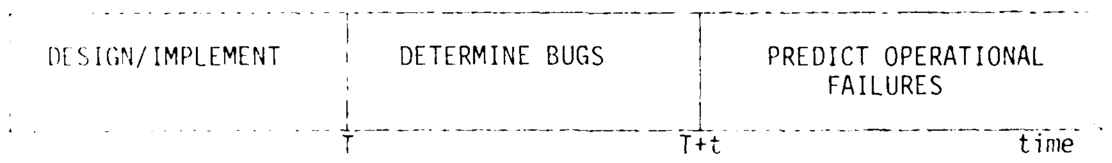
$$r(t) = a\phi(at-t_0)/\Phi(at_0).$$

Renewal theory. A common application for the reliability theory sketched above is in the prediction of cost of operation as the system functions within some prescribed maintenance policy. By knowing statistical properties of operational periods (e.g., such statistics as mean time between failure) and repair periods (e.g., replacing a failed component reduces the number of failed components by one) a variety of useful system parameters can be calculated. In concert with system design and maintenance policies (for example, survivability may be insured by redundancy and repair effort reduced by keeping a stock of spare parts) standard optimization models can be used to minimize total operating costs due to failure. The mathematics of the optimization techniques dictates very simple statistical models for these applications. Among the simplifications which seem to be essential one frequently finds:

1. failures are immediately detectable and attributable to a single component,
2. during periods of reduced operation the remaining load is shared by the rest of the system,
3. the renewed system and its predecessor have identically distributed failures.

3. SOFTWARE MODELS

The descriptions of extant software models presented here are obtained from the DACS summaries [7]. Although a wide variety of models are covered in the DACS extracts, we will highlight those which fall within the time-dependent reliability models of the traditional theory. The following developmental chart illustrates the use of these models.



At time T in the lifecycle of the software system, the statistical model is determined, usually by controlled observations of system errors (the "failures" of the traditional theory). In the interval [T,T+t] enough data is to be gathered to determine a failure rate $r(t)$. This coupled with a variety of subjective evidence concerning the behavior of the system in operation then determines an estimate of the underlying failure distribution. The predictive model is then applied in the interval [T+t,T+t+k], where k is a "regenerative" time less than the operational lifetime of the system.

The following table summarizes the utilization of the traditional models summarized in [7].

Distribution	Number of Models
Exponential (incl. geometric and Poisson)	7
Rayleigh or Weibull	3
Normal	1

As described above, the Exponential-Weibull distributions describe system failures when the system components are stressed materials. While there may be a body of experimental data which fits the appropriate failure rate functions, the assumption of one of these distributions as the underlying reliability distribution forces one or more of a number of questionable assumptions concerning errors in software. For example:

1. The number of initial program errors can be reliably estimated.
2. Error detection rate (failure rate) is proportional to the number of remaining errors.
3. Errors are discovered one at a time.
4. Once an error is detected, it can be located and removed immediately.
5. Error occurrence rate is constant.
6. Removing an error reduces the total number of errors by one.
7. Error occurrences are statistically independent.
8. The distribution of program inputs is known.
9. Error detection rate is proportional to the debugging effort.
10. Program size is constant over the lifetime of the program.

The exponential evidence to support these assumptions is contradictory [8]. Common sense would suggest that statistical models intended to describe physical systems would be ill-adapted to software, but it is still possible that nature allows an aggregate description of the error occurrence rate in typical software that is useful in practice. One test of this is careful experimentation on the assumptions 1-10. Allen Acree of the Georgia Tech Testing Laboratory has, for instance, gathered extensive data [9] on failure rates as related to the number of remaining errors in small programs (50-1000 lines) and found that it does support the exponential rate. In other studies by Acree and Timothy Budd of the Yale Testing Laboratory, however, [9,10] there is rather clear evidence that the strategic assumptions of independence of errors, single error occurrence, and immediate removal fail drastically in even moderately large systems. There is also considerable evidence that for large systems, most remaining errors lie in unexecuted portions of code, which means that failure rates cannot depend on either number of remaining errors or debugging effort for these systems.

Certain of the remaining assumptions also appear hard to confirm. For example, there is no methodologically acceptable way of estimated initial error percentages. The most commonly proposed technique is the error seeding procedure described in [11], which is based on the population statistics calculations described in Feller's classic text [12]. While this procedure gives the best unbiased estimator for certain random populations, it can badly overestimate or underestimate the number of errors when there is nonrandom mixing of seeded and natural errors. There is experimental evidence that such nonrandom mixing must take place [9,10].

Finally, the use of the Rayleigh or Weibull distributions to model programs modules presents severe mathematical difficulties for macro-modelling, since these distributions are nonreproductive; that is, overall system description is not a simple aggregate of module descriptions.

The usual validation of these models is the posterior observation of failure rates and costs [14]. The experimental technique is deficient and lacks some essential size constraints to allow statistical sampling techniques to be used with acceptable confidence. Studies aimed at validating these models have led to even more confusion:

> Basically, the results of the goodness-of-fit tests and other aspects of the data analysis indicate that none of the models fit very well...the various data sets showed obvious tendencies to increasing (in time) error rates: more or less contrary to the model assumption that mean error rates should be nonincreasing. It also seems clear to us that the observed increasing error rate situation cannot simply be written off as due to the introduction of errors during the debugging process or due to chance...the data sets themselves have some problems of a nature unknown to us...with better data more fits might have been obtained.[14;p.5-61].

Another set of difficulties has been clearly enunciated by Littlewood (see [15]): "we should be extremely careful of replacing the wealth of information in a probability distribution with simple summaries - whether these be parameters or moments of the distributions." In a sense, Littlewood's criticisms are even more fundamental than those attained above since they go to the heart of the matter - the correct interpretion of the statistical theory.

Finally, the use of the Rayleigh or Weibull distributions to model programs modules presents severe mathematical difficulties for macro-modelling, since these distributions are nonreproductive; that is, overall system description is not a simple aggregate of module descriptions.

The usual validation of these models is the posterior observation of failure rates and costs [14]. The experimental technique is deficient and lacks some essential size constraints to allow statistical sampling techniques to be used with acceptable confidence. Studies aimed at validating these models have led to even more confusion:

> Basically, the results of the goodness-of-fit tests and other aspects of the data analysis indicate that none of the models fit very well...the various data sets showed obvious tendencies to increasing (in time) error rates: more or less contrary to the model assumption that mean error rates should be nonincreasing. It also seems clear to us that the observed increasing error rate situation cannot simply be written off as due to the <u>introduction</u> of errors during the debugging process or due to chance...the data sets themselves have some problems of a nature unknown to us...with better data more fits might have been obtained.[14;p.5-61].

Another set of difficulties has been clearly enunciated by Littlewood (see [15]): "we should be extremely careful of replacing the wealth of information in a probability distribution with simple summaries - whether these be parameters or moments of the distributions." In a sense, Littlewood's criticisms are even more fundamental than those attained above since they go to the heart of the matter - the correct interpretion of the statistical theory.

If, indeed, the classical theory can be imported to software reliability there must be considerable attention paid to deriving relevant distributions from first principles and to developing the appropriate experimental and data gathering [12] techniques. It should be noted, however, that software errors must be viewed as design errors [8] (since there is no material wearing or stressing) and classical reliability theory does not deal satisfactorily with design errors.

Whatever the final resolution of these difficulties, it seems to us that there is no way around them. Estimations of reliability based on incorrect models, statistics improperly applied, or fuzzy notions of "error" can have unboundedly pernicious behavior - if the model doesn't fit there is simply no way to tell!

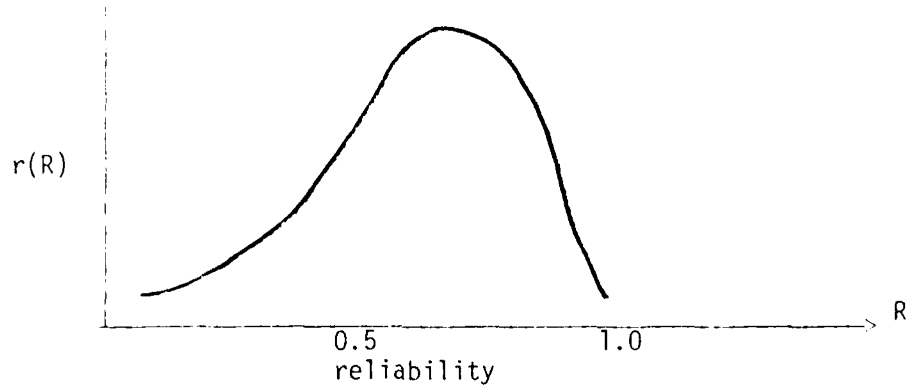
4. CONFIDENCE ESTIMATION

In a sense, the goals of traditional reliability theory and software reliability measurement are fundamentally at odds with each other. We have little opportunity to create large populations of identical systems and observe failure rates to obtain statistically meaningful measurements of failure distributions as indications of relative frequencies.

The more common situation is that a fixed programming-testing-validation method is imposed with a certain confidence level. The actual reliability of the system is then intertwined with this estimate of the system's reliability (since we stop testing when our estimate is satisfied).

We should then ask for experiments to confirm the level of confidence in the chosen methodology. In traditional statistical approaches to reliability, such objectively obtained parameters as number of error observed per unit time are used to infer the appropriate distributions.

We suggest that the observable parameters represent instead a varying quantity which represents a prior estimate of reliability. That is, we use a function r(R) to represent our level of confidence in the reliability level R of the software. For example, a program that has been subjected to mutation analysis [13] might be described by a function r with the shape shown below.



It is important to note that r(R) is merely a quantitatively assessed judgement; it is not necessarily an objective probability obtained by classical techniques. The r function may, however, be based on solid evidence and may form the basis for a scientific hypothesis concerning the long run behavior of a system. By applying Bayes' theorem [11], it is possible to gather "hard" evidence to support or deny the estimate of confidence r(R). From now on, we treat R as a random variable with probability density function r(R).

Suppose that in an experiment designed to refute a given reliability level R we make a large number of observations and gather a statistic, say, x, which is distributed as $E(x|R)$ for a given level R. Think of E as a distribution of errors for a given reliability; this is a quantity which can be observed and inferred in a given programming environment. The joint distribution for x and R is F(x,R). Let the associated pdf's be e and f, so that

$$f(x,R) = r(R)e(x|R),$$

the marginal density for x is:

$$g(x) = \int_R r(R)e(x|R)dR.$$

Then the conditional distribution for R is given by the density

$$h(R|x) = r(R)e(x|R)/g(x),$$

which represents our level of confidence in R given the results of the experiment yielding x.

Even though these results merely restate Bayes' theorem, the interpretation implied by the results of the experiment yielding x is not necessarily subjective. The key is rather in the degree of objectivity with which r(R) is obtained.*

At the heart of this treatment of software reliability is another view of the role of statistical statements. This point of view rejects the idea that a meaningful probability of correct operation can be assigned to a piece of software. Rather, any such assessment must be interpreted as a "level of confidence" in the process used to validate the program. This effectively shifts the statistical burden from the program to the methodology used to produce and validate it. One problem with the traditional approach, which we noted briefly above, is that there is no acceptable sense in which the frequency interpretation of the reliability function R(t) can apply to software; there is, for example, no reasonable application of the

*cf. Feller's criticism of Bayesian analysis in reliability [12, p.124].

law of large numbers to obtain failure rates. This view of probability -- the limit of a physically observable relative frequency -- is not the only view. It is possible to construct alternative subjective probabilities. In fact, James Bernoulli in 1713 described probabilities as "degrees of confidence."

Even though we use the term "subjective" in describing the non-frequency version of R(t), it does not follow that the measure represents an *ad hoc* assessment of reliability. For example, the prior distribution may not really represent a subjective estimate of reliability. It may be the result of experimental research and summarization. This suggests an iterative procedure for "updating" the state of knowledge regarding the reliability of the validation methodology.

Littlewood [15] also adopts the Bayesian approach - with an important extension to the software design phase. For essentially the reasons outlined above Littlewood also rejects the frequency interpretation of R(t). He goes farther, however, in exploiting the ability of Bayes' theorem to update previous reliability measurements in light of new data: "periods of failure-free working cause the reliability [i.e., confidence] to improve...it is interesting to compare this with the [traditional methods in which]...reliability improvements can only take place at a failure, since it is only at such a time that an error can be removed." [15]. This, Littlewood cites as a contradiction to the common sense approach to estimating software confidence - namely, that there is no a priori reason that confidence should increase as failures occur.

In other words, an assessment of reliability represents a judgement relative to a prior reference "standard" which the user presumably understands, believes, and against which he is being required to wager.

A useful reliability theory then should incorporate notions which are so far untouched. For example, an economic theory that develops a concept of utility for software would be extremely helpful. Hence, for the reliability measurement to be useful, it must only order uncertain events so that the user may consistently win when he adopts a rational betting strategy [16].

REFERENCES

[1] Z. Manna, Mathematical Theory of Computation, McGraw Hill, 1974.

[2] S. Gerhart and J. Goodenough, "Toward a Theory of Test Data Selection," in R. Yeh (Editor), Current Trends in Programming Methodology: Vol. 2, Program Validation, Prentice Hall, 1977, pp. 44-79.

[3] E. Dijkstra, A Discipline of Programming, Prentice Hall, 1977.

[4] K.C. Kapur and L.R. Lambertson, Reliability in Engineering Design, Wiley, 1977.

[5] M. Zelen (Editor), Statistical Theory of Reliability, University of Wisconsin Press, 1964.

[6] R.E. Barlow and F. Proschan, Mathematical Theory of Reliability, Wiley, 1965.

[7] DACS, Quantititive Software Models, March 1979.

[8] R. Longbottom, Computer System Reliability, Wiley, 1980.

[9] A.T. Acree, "On Mutation," PhD Thesis, School of Information and Computer Science, Georgia Institute of Technology, June 1980.

[10] T.A. Budd, "Mutation Analysis of Program Test Data," PhD Thesis, Department of Computer Science, Yale University, June 1980.

[11] T. Gilb, Software Metrics, Prentice Hall, 1978.

[12] W. Feller, An Introduction to Probability Theory and Its Applications, Vol. 1, Wiley, 1968.

[13] A.T. Acree, T.A. Budd, R.A. DeMillo, R.J. Lipton, and F.G. Sayward, "Mutation Analysis," Report GIT-ICS-79/08, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia 30332.

[14] R.Schafer, et.al., "Validation of Software Reliability Models," RADC-TR-79-147, Rome Air Development Center, June 1979.

[15] B. Littlewood, "Software Reliability Measurement, Some Criticisms and Suggestions,", Presented at the Software Life Cycle Management Workshop, 21-23, August 1977, pp. 473-488.

[16] M. DeGroot, Optimal Statistical Decisions, McGraw-Hille, 1970.

Statistical Models of Software Reliability

Rich DeMillo and Fred Sayward

Martin Shooman, Polytechnic Institute of NY: You mentioned that some of the people working in the reliability modelling area compiled data that showed that some modelling assumptions were false and had tried to explain this away by finding fault with the data. Can you cite some of the data? I'm not aware of any such studies.

Rich DeMillo: I don't remember the title of the report, but it was a June 1979 RADC report on validating reliability models. I'll be happy to send you the citation.

Shooman: Do you know the author?

DeMillo: No, no, I really don't know the author.*

Shooman: I've worked in this area for over ten years and I don't know that that's so, so I'm a little puzzled...

DeMillo: (Assenting noise.)

Fred Sayward: One thing I might mention is that in all of the material I've seen in this area, there has been very little work done on using standard statistical tests to validate distributional assumptions. That seems to be a missing ingredient, and there are standard tests, like the chi square test and tests of that nature.

DeMillo: The Rome study analyzed a large mass of reliability data, some of it from internal Hughes Aircraft projects. The deficiencies of the data are discussed in the paper.

Shooman: Yeah, I think you're talking about a study that was done at

*The report is RADC Report TR-79-147, "Validation of Software Reliability Models" by R.E. Schafer, J.E. Angus, J.F. Alter, and S.E. Emoto, of Hughes Aircraft.

Rome by Allan Soo[illegible]. A lot of the data did not have basic parameters on time when certain failures occurred and other necessary data. These are necessary for some of the models which John Musa calls execution-time models. These models have been validated by other sets of data. Also, rather than make statistical tests, some people have made estimates. Then they have made measurements later when the software was placed into use. The estimates agreed to within twenty, thirty percent of the actual usage measurements. That's a rather strong proof that there is some validity to the model. One of the big problems in this as in other areas is that, it's very difficult and costly to gather data. So, rather than, say that the models are invalid because they haven't been validated, I think you should look at some of the studies where they have been validated.

Sayward: Any further questions, or comments?

Merv Muller, World Bank: It's a comment. I think I've made it before to the panel. I get very uncomfortable when I hear people who are testing models using chi squared tests. The problem of chi squared is you don't know what the alternative is. It's every other possible point in the parameter space of all possible models that you could ever have thought of. It's obvious if you don't take very much data you will most likely accept the fact that you don't have enough data, or you might accept a model and you could take too much data. I don't care what the model is, you'll reject it. All you need to do is go back and look at the Rand studies in the late 1940's on trying to test randomization. They put out a big thick book, and they ran many, many tests. What they really came down to on the bottom line was they couldn't come to any conclusion because the theory is lacking. And others have tried to adjust the chi square test for the fact that you have an interval of possibilities. If you insist that there's a single point (I don't care what distribution it is) you're ultimately going to reject it because you don't know what the power of the test is: you don't know what you're comparing it to. I think it would be a pity if we got hung up over a statistical test that we've developed for a certain class of statistical analysis. I like what Shooman said. I think the purpose of using data is ·ɔt to test, but to try to estimate something...whatever that may be, because you're interested in the future. I may have worn out my welcome on this, but there is an inappropriate use of statistical techniques, and I don't know whether they answer the question of the validity of the model or not, but I would urge that people pay more attention to statistical theory.

DeMillo: Let me ask a question of Martin Shooman, as long as he's here. It seems to me that the general consensus among people who work in engineering reliability theory is that the theory is just not well adapted to design errors. Is it your contention that the errors that you find in programs are not design errors, that they're somehow equivalent to wear-out errors? Failure errors?

Shooman: Let me address the first statement. Each year at the annual Reliability and Maintainability Symposium you see about, between seven hundred and a thousand of these people. I regularly attend these symposia and I don't find that to be true.

DeMillo: I'm talking now about the time-dependent theory.

Shooman The general consensus among these people is not that this doesn't, all right? What you find is that when you deal in failure of any type you find various modes of failure. It is true that in any failure of hardware the predominant mode of failure is wear-out, an actual physical failure of the device. Although perhaps you still find, perhaps ten percent of the modes of failure are design errors. When you deal with software,the predominant modes of failure are design errors. This in no way invalidates the basic approach. All it means is that you end up making different models. True, your distributions may look different. But, in general, you're dealing with a very simplistic model. It's somewhat invariant to the distribution. People in a first-order model trying to fit data can fit an exponential model, can fit a gamma model, and so on. Sure, you may get errors of ten or twenty or thirty percent, but the techniques for estimating the parameters of the models are no more accurate than twenty or thirty percent, so the distribution is really not as important as how you're going to measure the parameters of the model. The measurement techniques are crude enough so that even using a one-parameter rather than a two-parameter model won't very much make any difference in the results. The fact that software doesn't wear out is well known to all modellers and they take this into account and they do not use models which are based upon software wearing out.

Sayward: OK. Would anyone else like to make a comment or have a question?

Irwin Nathan, Xerox: The problem is that you have a very complex system, all kinds of subsystems, routines, and human interfacing, and you really can't find out what's going on. All you know is that at certain intervals, which are randomly distributed, or distributed according to some probability distribution, that you're going to get failures, and you can forecast that, and as you keep improving, and finding these design anomalies, and correcting the failure rate gets lower and lower. You eventually get to the point where you can tolerate it, OK? That's essentially what you do in hardware systems. Tolerate it. There are a number of design errors, and there are a number of components which are probably exceeding some design limit, in some way, and there's a few components that are wearing out. And you add all these things up, and you take a macro look at it, and you say, it's a constant failure rate (or whatever you have). And the same thing is probably true with software, based on what we have done up to date. And I think that's where the situation is as far as software reliability is going: to try and estimate those parameters. At least you know where you are, and you can have some statement of probability of success for a system which is going to operate and evolve. It's only when you get to the complex programs that it counts.

DeMillo: The communities that are involved in reliability estimation and in program testing should have a lot to say to each other. The assumptions that I chose for this slide (pause) have been addressed by a number of studies in program testing. I gather from your comments and Shooman's comments, that in reliability estimation you take a very summary view of the failure process. In program testing we take a microscopic view of the process. The observations that we get through program testing are fundamentally at odds with the assumptions that are built into these reliability models.

Voice: For instance, take the assumption of statistically independent errors. Or constant program size.

Sayward: I was going to add that one common thing you find in program testing is that most of the remaining errors are in code that hasn't been executed -- or even, in some cases, can't be executed, except under very bizarre conditions. That's certainly not a continuous type of thing.

DeMillo: I guess what I'm doing is suggesting a number of experiments. The models, these time-dependent models, rise or fall on a given set of assumptions, and I'm saying that those assumptions can be dealt with empirically. Now, if we don't have adequate data to determine the validity of an assumption, I think that we should put it together.